PRUNING-BASED REINFORCEMENT LEARNING FOR COMPRESSION OPERATOR COMPOSITION WITH ADJUSTABLE OBJECTIVE WEIGHTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Compression algorithm selection depends on application preferences over compression rate, compression time, and decompression time. We therefore use an adjustable weighted objective combining compression rate, compression time, and decompression time. Compression algorithms share operators that either reduce data size or transform data to facilitate compression. We identify and recombine these operators to seek a lower objective value than fixed operator pipelines. We propose operator-composition reinforcement learning (OCRL). It combines group relative policy optimization (GRPO) with pruning. The policy learns from measured pipeline costs. A byte lower bound prunes branches that cannot improve the best verified candidate. OCRL recommends a compression operator pipeline for user-specified objective weights. It seeks a lower objective value than the compared baselines on real-world datasets.

1 INTRODUCTION

Lossless compression serves applications with different requirements for storage, transmission, and data processing. For long-term storage with limited capacity, a low compression rate reduces the space needed to retain data. Examples include data lakes and astronomical archives (Kuschewski et al., 2023; Truong et al., 2025). High-frequency industrial monitoring instead requires fast compression to keep pace with incoming sensor measurements (Wang et al., 2023; Tang et al., 2025). Consider sensors that continuously monitor steelmaking equipment. If compression cannot keep pace, pending measurements accumulate in buffers. Sustained overload can cause data loss when buffers are finite and the source cannot be slowed. For local-to-cloud synchronization, compression time and compression rate jointly affect synchronization latency (Zhu et al., 2025). Fast compression reduces processing delay, while smaller outputs reduce transmission volume. Bandwidth constraints also motivate compression for robotic tactile data (Cheng et al., 2026). For analytical queries that repeatedly decode compressed columns, fast decompression reduces the time spent decoding each scan (Kuschewski et al., 2023; Afroozeh et al., 2023). A low compression rate can also reduce data transfer from storage to memory. Model-weight compression has a related trade-off between memory footprint and decoding throughput (Yang et al., 2026). Algorithm selection therefore depends on how the application prioritizes these three costs.

1.1 MOTIVATION

Pipeline selection depends on the application objective, the operators available for composition, and the input data.

Different application requirements make compression algorithm selection difficult. Storage-limited archives prioritize smaller representations. High-rate ingestion prioritizes compression speed, whereas repeated analysis prioritizes decompression speed. The compression rate ρ is complete compressed bytes divided by original numeric bytes; throughout this paper, it denotes a dimensionless size fraction. Lower values are better. Figures 1(a) and 1(b) show reported compression rate against compression and decompression time, respectively. The archived results illustrate the

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

trade-offs; Appendix A gives the selection and ranking rules. Thus, a size ranking alone does not determine the preferred algorithm across these settings.

A fixed compression operator pipeline is suboptimal. An operator performs one data transformation or encoding step. Delta represents changes between values, while bit packing stores small integers using fewer bits. A composition connects operators so that each step changes the data available to the next. A fixed pipeline restricts these choices and may retain stages whose size reduction does not justify their processing cost. Recombining shared operators can expose useful regularity or avoid unnecessary work for the chosen objective. However, each stage introduces choices of operator, parameters, and how to process its output data flows. These choices create a large search space that grows rapidly with depth, making exhaustive evaluation costly.

The preferred operator pipeline depends on the dataset. The input determines which regularities operators can exploit. Slowly changing values can benefit from small residuals, whereas repeated values can benefit from run-length representations. The same transformation can increase storage or processing cost when these regularities are absent. Thus, a pipeline selected for one dataset need not suit another, even under the same objective weights. This dependence motivates pipeline selection conditioned on both input features and application preferences.

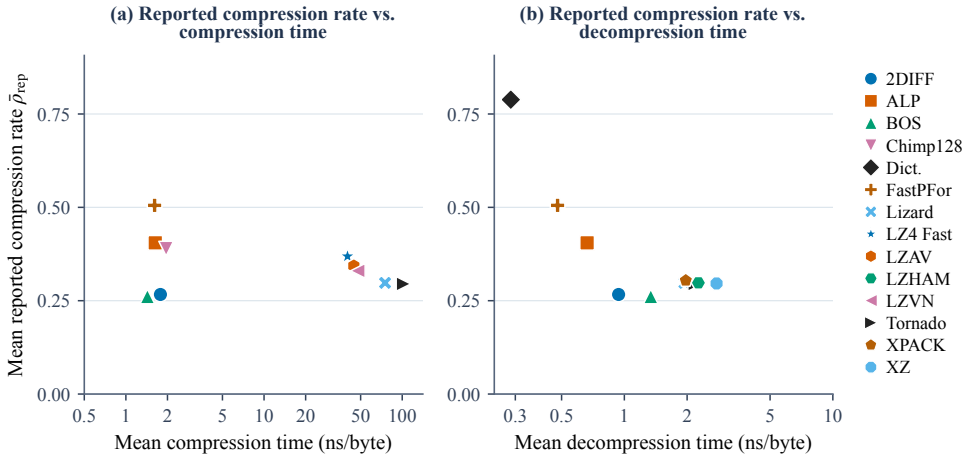


Figure 1: Figures (a) and (b) show separate Pareto-ranked Top-10 selections from archived method labels, each averaged across the same dataset collection. Reported sizes are not audited complete frames, archived timing protocols differ, and Bird-migration uses only two integer columns (Appendix A).

1.2 Contribution

We propose OCRL, a pruning-based GRPO algorithm that recommends reversible operator pipelines from input features and objective weights. Its three contributions address the preceding sources of variation.

(1) We make the compression objective adjustable to application requirements. User-specified weights set the relative priority of compressed size, compression time, and decompression time.

(2) We search reversible operator compositions with certified pruning and GRPO. A byte lower bound safely prunes candidates, while GRPO searches the remaining legal pipelines using measured objective values. The weight-dependent pruning rule in Propositions 4.1 and 4.2 avoids full evaluations of branches that cannot beat the measured pipeline.

(3) We make pipeline recommendation data- and preference-conditioned. One frozen policy uses input features and objective weights to guide budgeted search on held-out data.

2 Related Work

Lossless compression relies on matching a data representation and coding mechanism to the structure of its input. Learned lossless compression has used bits-back coding with latent variables and probabilistic circuits (Townsend et al., 2019; Liu et al., 2022). AstroCompress compares neural and non-neural lossless methods across astronomical imaging modalities (Truong et al., 2025). TaCo benchmarks compression on heterogeneous tactile data (Cheng et al., 2026). ECF8 exploits exponent statistics in model weights (Yang et al., 2026). These studies illustrate the diversity of useful compression mechanisms. We focus on selecting and composing reversible operators for numerical columns under adjustable size and time objectives.

2.1 Compression Algorithms

We group the compression-method labels considered in this study, including implementation variants, by their dominant compression mechanism. *Residual-based* methods, including Sprintz, FastPFOR, and ALP, exploit prediction or reference-based representations to reduce numerical width (Blalock et al., 2018; Lemire & Boytsov, 2015; Afroozeh et al., 2023). *Reordering-based* methods expose locality through block sorting and subsequent move-to-front recoding (Burrows & Wheeler, 1994; Bentley et al., 1986). *Match-based* methods, from Lempel–Ziv (LZ) to DEFLATE, Brotli, and Zstandard, replace repeated substrings with references (Ziv & Lempel, 1977; Deutsch, 1996; Alakuijala & Szabadka, 2016; Collet, 2021). *Coding-based* methods, including Huffman, arithmetic coding, asymmetric numeral systems (ANS), and Simple8b, exploit symbol probabilities or integer widths (Huffman, 1952; Witten et al., 1987; Duda, 2013; Anh & Moffat, 2010). *XOR-based* methods use bitwise exclusive OR (XOR) to exploit similarities between observed or predicted floating-point bit patterns. Examples include Gorilla, FPC, Chimp, and Elf (Pelkonen et al., 2015; Burtscher & Ratanaworabhan, 2009; Liakos et al., 2022; Li et al., 2023).

2.2 Compression Operators

The compression algorithms above share operator-level building blocks (Deutsch, 1996; Pelkonen et al., 2015; Afroozeh et al., 2023). We organize them into eight operator groups: *Bit processing*, *Layout and state*, *LZ decisions and format*, *Match and repetition*, *Numeric representation*, *Probability coding*, *Reordering and symbol recoding*, and *Residual formation*. Appendix C provides the complete operator catalog and reversible examples.

2.3 Learning and Heuristic Search for Compression Composition

Automatic composition has been explored through sampling, evolutionary search, and reinforcement learning. BtrBlocks recursively selects encodings using sample-based compression-rate estimates (Kuschewski et al., 2023). AdaptiveFC uses a genetic algorithm to search pipelines of fine-grained operators for floating-point files (Rodriguez et al., 2024). DICOMP uses reinforcement learning to explore arithmetic-operator sequences for integer compression (Farhat et al., 2025). AMMMO (Adaptive Multi-Model Middle-Out) uses reinforcement learning to set compression-mode parameters for a subsequence (Yu et al., 2020). It then chooses among the resulting encodings for each point. These compression studies search over encodings, operators, or their parameters. Outside compression, preference-driven reinforcement learning has trained a single policy across objective weights (Basaklar et al., 2023). GRPO was introduced for mathematical reasoning as a PPO variant (Shao et al., 2024; Schulman et al., 2017). Our formulation uses reinforcement learning to construct rooted, ordered operator trees under type and reversibility constraints. The objective combines storage, compression time, and decompression time. The search selects operators, their execution order and parameters, and the encoding operators for individual branches.

3 PROBLEM FORMULATION

Section 3.1 defines operators, typed data flows, and the tree structure of their compositions. Section 3.2 then defines complete reversible pipelines, the three evaluation metrics, and the weighted minimization problem.

3.1 OPERATORS, DATA FLOWS, AND COMPOSITION

Let $\mathcal{O} = \{O_i\}_{i=1}^{30}$ denote the operators in the eight groups named in Section 2.2. The index i identifies an operator, not an implementation. Appendix C lists their identifiers, mechanisms, and reversible examples. A data flow is $F = (\tau, n, \mathbf{v}, \beta)$. Its components record the representation type, element count, ordered values, and representation width. Floating-point values retain their original bits. The input x is carried by the root data flow $F_{0,1}$.

An operator pipeline p forms a rooted, ordered tree $\mathcal{T}_p$. We write $\mathcal{T}_t$ for the partial tree at construction step t. Each non-root node has one parent, and independently extended branches do not rejoin. At step t, operator O_{i_t} consumes an ordered tuple $\mathcal{I}_t$ of unconsumed outputs from one parent and produces

$$(\mathcal{F}_{t+1}, h_{t+1}, m_t) = O_{i_t}^{\vartheta_t}(\mathcal{I}_t; h_t). \tag{1}$$

Here ϑ_t selects the implementation and parameters, while h_t is the execution context. The output tuple is $\mathcal{F}_{t+1}$, and m_t is the reconstruction information. Each output is identified by its producing node v and port q as $F_{v,q}$. An ordinary operator increases path depth by one, up to K.

The routing controls $\mathcal{S} = \{\text{STOP}, \text{SPLIT}, \text{CONCAT}\}$ are not operators. STOP marks one data flow as stopped. SPLIT retains continuous-partition boundaries and lets each branch continue independently. CONCAT joins compatible outputs of one parent in recorded order. Continuous SPLIT differs from Column Split O_6. Each control consumes one of the H decisions. The pipeline is complete only when all data flows are stopped. Figure 2 illustrates the choices.

3.2 REVERSIBLE PIPELINES AND THE OPTIMIZATION OBJECTIVE

For nonempty numeric data x and a legal pipeline p, let Enc_p and Dec_p denote its encoding and decoding procedures. The complete compressed frame is $C_p(x) = \mathrm{Enc}_p(x)$ and must satisfy $\mathrm{Dec}_p(C_p(x)) = x$ bitwise. The frame includes every encoded leaf, the operator tree $\mathcal{T}_p$, and all reconstruction records m_t required to reverse its steps. Let $\mathcal{P}_{\mathrm{legal}}(x)$ be the finite set of legal pipelines for x. Each pipeline must satisfy the fixed tree constraints, operator and parameter choices, maximum path depth K, and decision budget H.

Let $|x|$ denote the original numeric byte count before any transform. Let T_E, T_D denote the complete-frame encoding and decoding times in nanoseconds. The same byte-count notation gives the complete compressed size $|C_p(x)|$. Let E, D denote encoding and decoding time per original byte. Their normalized costs are $\widehat{E}, \widehat{D}$, with positive reference times E_0, D_0 in the same units. The three lower-is-better metrics and normalized time costs are

$$\rho = \frac{|C_p(x)|}{|x|}, \quad E = \frac{T_E}{|x|}, \quad D = \frac{T_D}{|x|}, \quad \widehat{E} = \frac{E}{E + E_0}, \quad \widehat{D} = \frac{D}{D + D_0}. \tag{2}$$

The positive references E_0, D_0 are fixed per input and measurement protocol before search. All methods and weights share these references. Normalization makes the time terms dimensionless and bounded while preserving their ordering. Raw E, D are also reported in ns/byte. For weights $w = (a, b, c)$ with $a, b, c \geq 0$ and $a + b + c = 1$, define

$$J_w(x, p) = a\rho + b\widehat{E} + c\widehat{D}. \tag{3}$$

We write J for $J_w(x, p)$ when the input, weights, and pipeline are clear. Terminal rewards and final selection both use this cost.

Problem 1 (Objective-aware operator composition)**.** Given x, w, and the fixed measurement protocol, find an optimal legal pipeline $p^\star(x, w)$ such that

$$p^\star(x, w) \in \operatorname*{arg\,min}_{p \in \mathcal{P}_{\mathrm{legal}}(x)} J_w(x, p). \tag{4}$$

The optimum is relative to the declared finite space of reversible pipelines.

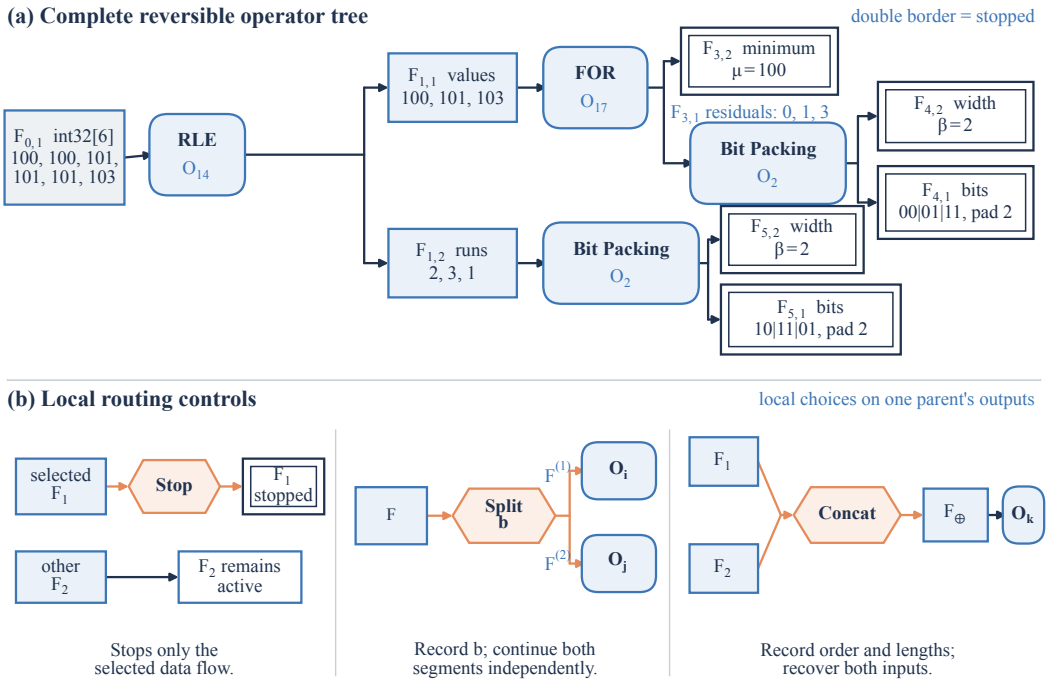


Figure 2: A complete reversible operator tree with local routing choices on one parent's output tuple. SPLIT extends branches independently, while CONCAT joins outputs before branch extension. In Figure (a), μ is the FOR minimum, β the packing width, and `pad` the number of padding bits. Double-bordered boxes are stopped data flows. In Figure (b), $F_{\oplus}$ is the data flow produced by CONCAT. The serialized output retains every stopped leaf and all reconstruction metadata.

4 PRUNING-BASED GRPO

Exhaustively evaluating all legal combinations of operator implementations requires repeated compression and decompression for each input. The resulting cost makes it impractical to construct labels identifying the optimal pipeline for each input and set of objective weights. We therefore use GRPO post-training without labeled optimal pipelines. The policy learns from rewards computed using measured compression rate, compression time, and decompression time. Section 4.1 defines the construction state, policy actions, and terminal rewards. Section 4.2 presents the pruning bound and its use in GRPO sampling and updates.

4.1 CONSTRUCTION STATE, ACTIONS, AND REWARDS

We use one preference-conditioned policy over the data-flow state defined in Section 3.1. Let $\mathcal{A}_t$ be the ordered active data flows and $\mathcal{L}_t$ the stopped leaf data flows at step t. Every member is an $F_{v,q} = (\tau, n, \mathbf{v}, \beta)$ produced at an identified tree node and output port. The construction state s_t contains the current operator tree $\mathcal{T}_t$, these data flows, their reconstruction context, weights w, and remaining budget $H - t$. The deterministic feature map ϕ encodes this state as a numerical model input. The policy π_θ has trainable parameters θ, and u_t is its next sampled action. The root data flow $F_{0,1}$ and weights w initialize s_0. At each step, the policy samples an action:

$$u_t \sim \pi_\theta(\cdot \mid \phi(s_t)) . \tag{5}$$

Equation 5 makes the model input a feature encoding of the current F-based state, rather than raw x or a bare byte string. GRPO updates the policy parameters from terminal rewards.

The feature map $\phi(s_t)$ includes w, the remaining budget, and each active data flow's producer operator, type, element count, width, and current byte count. The same w defines the terminal cost

$J_w(x, p)$. Thus, identical partial operator trees can receive different action probabilities under different weights while sharing θ.

An action selects input data flows from one parent, an operator or routing control, and its parameters. Tree structure, type, and budget masks remove illegal choices. A mask assigns zero sampling probability to a disallowed action. A terminal pipeline is accepted only after complete-frame encoding and independent bitwise recovery of the input.

Applying u_t updates the selected portion of s_t. An ordinary operator consumes its input data flows and places its typed outputs in $\mathcal{A}_{t+1}$. STOP moves a selected data flow to $\mathcal{L}_{t+1}$. SPLIT creates contiguous child data flows, while CONCAT joins compatible outputs of one parent before branch extension. Untouched data flows and reconstruction records persist in the next state. These local updates preserve the ordered-tree constraint.

Let $R_w(x, p)$ denote the terminal reward of a verified pipeline. It is $R_w(x, p) = -J_w(x, p)$, while nonterminal rewards are zero. A deterministic completion stores the current active payloads unchanged and stops every remaining data flow. When this completion is legal within the remaining budget, its measured cost J_t is included as a state feature. Otherwise, $J_t = \perp$, where $\perp$ marks an unavailable value. This feasible completion gives an upper bound on the best reachable cost. It does not give a lower bound for pruning.

Only stopped, reversible pipelines provide objective measurements for training. An intermediate data-flow size is a state observation, not the compressed size $|C_p(x)|$. The complete-frame measurement includes every stopped leaf, the operator tree, and reconstruction records. Its byte count and full encoding and decoding times determine ρ, E, and D. A failed bitwise round trip yields no terminal reward or policy update.

4.2 OBJECTIVE-BOUND PRUNING AND GRPO UPDATES

Safe objective-bound pruning. A large intermediate payload need not imply a poor final pipeline: later operators can still compress it. We instead count unavoidable bytes. Fix a nonempty input x, its finite legal search space, and the complete-frame format. For a partial state s, let p_s be its prefix and $\mathcal{Q}(s)$ the legal action sequences ξ that stop every active data flow within the remaining budget. Let B_{fix} be the mandatory frame overhead in bytes. Let $B_{\text{keep}}(s)$ be the byte count of reconstruction records that every continuation must retain unchanged. These records are separate from fixed overhead and leaf payloads. Index the active data flows by ℓ, with operator depth d_ℓ. For each depth-K data flow, let b_ℓ count distinct payload bits certified to survive every remaining routing, STOP, and final serialization step. Counted bits from different data flows must be disjoint. Under an exact-bit leaf-frame contract, $b_\ell = n_\ell \beta_\ell$; otherwise $b_\ell = 0$ unless a separate disjoint retained-bit certificate is available. An ordinary operator increases depth by one and cannot extend a depth-K data flow; stopped data flows cannot be reactivated. Continuous SPLIT preserves payload bits and depth, while local CONCAT preserves payload bits and takes the maximum input depth. Let $L(s)$ be the unavoidable-byte bound and $\underline{J}_w(s)$ its corresponding objective bound.

$$L(s) = B_{\text{fix}} + B_{\text{keep}}(s) + \left\lceil \frac{\sum_{\ell:\, d_\ell = K} b_\ell}{8} \right\rceil, \qquad \underline{J}_w(s) = a \frac{L(s)}{|x|}. \tag{6}$$

Metadata that can still be rewritten or compressed and payloads below depth K are excluded from $L(s)$. Summing certified retained bits before byte rounding allows concatenation to share padding.

Proposition 4.1 (Lower bound from unavoidable bytes)**.** *Under the certified-retention conditions above, every $\xi \in \mathcal{Q}(s)$ satisfies $|C_{p_s\xi}(x)| \geq L(s)$. If $L(s)$ exceeds the size of a verified complete frame, no completion of s can improve its storage size.*

If $B_{\text{fix}} + B_{\text{keep}}(s)$ exceeds a verified frame size, it certifies pruning even if no data flow has reached depth K. When every active data flow reaches K, the bound also applies to continuations using only routing controls.

Proposition 4.2 (Lower-bound pruning for the weighted objective)**.** *Fix the normalization references and measurement protocol, and let $w = (a, b, c)$ have nonnegative weights. For every $\xi \in \mathcal{Q}(s)$, $J_w(x, p_s\xi) \geq \underline{J}_w(s)$. If U_J is the measured objective of a retained, verified complete pipeline for the same input and weights, then*

$$\underline{J}_w(s) > U_J \quad \Longrightarrow \quad \textit{discard every continuation of } s. \tag{7}$$

For example, if $|x| = 128$ bytes, the certified $L(s) = 96$ bytes, and $a = 0.4$, the objective bound is 0.300. An incumbent with $U_J = 0.200$ rules out this entire branch without evaluating its suffixes. Using 96 bytes of still-compressible intermediate payload would not justify the same decision.

The strict inequality preserves tied solutions. The incumbent also remains available after pruning. If only one optimum is required, equality may be pruned while retaining the incumbent, but tied optima may be lost. For $(a, b, c) = (1, 0, 0)$, the rule reduces to storage pruning. For $a = 0$, this storage-derived certificate gives no positive objective bound. It cannot prune on time alone. The guarantee applies only to the declared legal space and frame representation, not unsearched operator or parameter domains. Appendix B proves both propositions and separately states the conditions for storage-only dominance pruning.

Figure 3 shows training, validation, and frozen test-time inference. There, θ' denotes updated parameters, $\theta^\star$ is the validation-selected checkpoint, and $\hat{p}$ is the best verified test pipeline. Superscripts val and test identify validation and test inputs.

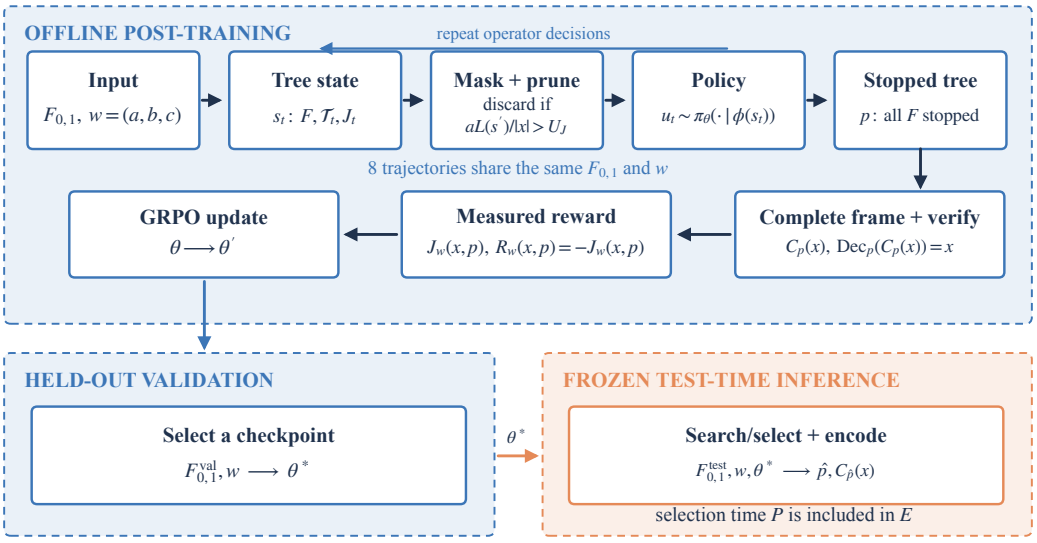


Figure 3: GRPO post-training and frozen inference. Structural masks and the certified bound $aL(s')/|x| > U_J$ filter successor states s' before sampling; complete, bitwise-verified frames supply rewards. Held-out validation selects the test-time checkpoint.

Pruned policy search. For each input and preference, the search retains the best verified pipeline and its cost U_J. Before action sampling, the controller materializes successors that satisfy the tree structure, type, and budget constraints. It then masks successors certified by (7). This work counts toward the search budget. The policy samples from the survivors, including uncertified prefixes regardless of their current size. Completed candidates are encoded, decoded, and measured. A trajectory is a sequence of states and actions from s_0 until every data flow is stopped or construction ends unsuccessfully. One sampling group contains eight trajectories constructed for the same input x and weights w. During training, the pruning threshold is fixed within each sampling group. Verified improvements tighten it for subsequent groups.

Each group compares eight trajectories under the same input x, weights w, and incumbent U_J. The fixed incumbent gives every trajectory the same pruning rule within the group. Group-relative normalization compares measured pipelines under one objective. A verified trajectory with lower $J_w(x, p)$ than its peers receives a higher relative advantage. The policy update uses this advantage without differentiating through compression or decompression.

Pruning changes the available continuations rather than assigning an estimated terminal reward. Pruned successors are never sampled and receive no reward. Incomplete groups produce no policy update. GRPO subtracts the within-group mean reward and scales by the within-group standard

deviation. This calculation gives relative advantages, which are set to zero when reward variation is negligible. These advantages update the shared policy through a clipped objective. The objective limits the incentive to move sampled-action probabilities too far from the policy that generated the group. At inference, policy parameters are frozen and the search returns the best verified candidate. Safe pruning preserves the minimum $J_w(x, p^\star(x, w))$ in Equation 4. A sampled pipeline is certified optimal only after all remaining candidates are evaluated or ruled out.

5 EXPERIMENTS

Section 5.1 specifies the data, methods, metrics, and selection protocol. Section 5.2 presents the six-weight comparison. Section 5.3 compares weight-conditioned selection with fixed routing. Table 1 records the available measurements. Online selection time P is included in compression time E when selection occurs during encoding.

5.1 EXPERIMENTAL SETUP

Data and scope. We report tests on five datasets from different application domains: rating, Comments, Arade, Bar-Crawl, and Telco. For each dataset, we retain a prefix of whole columns totaling at most 1 GiB. We split each column in row order into 10% training, 10% validation, and 80% test data. Each controlled campaign fixes the input columns, types, row ranges, and missing-value rules. Every valid complete frame must reconstruct the input bits. The reported subset contains 64 controlled test columns totaling 2,348,490,704 original numeric bytes. The subset was selected post hoc from cases favorable to the interim selector at $(0.4, 0.3, 0.3)$ under the original online-only J; it is not a full-dataset summary. The archival Best single row uses 64 whole columns rather than the controlled test split.

Methods and selection. We compare a retrospective single-algorithm reference, a deterministic weight-gated interim selector, and AMMMO. AMMMO uses a fixed, weight-independent configuration rather than retuning for each preference.

Metrics and preferences. We report ρ, compression time E, decompression time D, and selection or fitting cost P when available. All metrics are lower-is-better; times use ns per original numeric byte, except the marked offline fit cost, which uses training bytes. Table 1 evaluates six weights; online P is part of E, while offline P is added to E only for the diagnostic J with method-specific timing scope. The superscripts † and ‡ mark the retrospective whole-column reference and the two-version OCRL row, respectively.

5.2 RESULTS UNDER SIX OBJECTIVE WEIGHTS

The reported OCRL routes map input data and objective weights to complete compressed representations. OCRL has a lower displayed J than AMMMO under all six weights in Table 1. At $(0.4, 0.3, 0.3)$, OCRL records $(\rho, E, D) = (0.29, 7.71, 3.26)$, compared with $(0.52, 135.60, 6.36)$ for AMMMO; E and D are in ns/byte. OCRL conditions path selection on the weights and input type, whereas AMMMO retains a weight-independent physical configuration. This conditioning lets OCRL target the active terms of J instead of using one configuration for every objective.

Table 1: Post-hoc comparison of compression rate, selection or fitting cost, compression and decompression time, and weighted objective under six weight settings.

	$(1,0,0)$		$(0,1,0)$			$(0,0,1)$	
Method	ρ	J	P	E	J	D	J
Best single algorithm[†]	0.23	0.232	21481.81	0.65	0.999	0.23	0.031
OCRL[‡]	0.21	0.210	17.02	5.54	0.598	2.91	0.293
AMMMO	0.52	0.518	126.27	135.60	0.899	6.36	0.475

	$(0.5,0.5,0)$				$(0.5,0,0.5)$			$(0.4,0.3,0.3)$				
Method	ρ	P	E	J	ρ	D	J	ρ	P	E	D	J
Best single algorithm[†]	0.25	21481.81	0.84	0.625	0.25	0.63	0.164	0.25	21481.81	0.88	0.70	0.428
OCRL[‡]	0.29	17.02	7.83	0.454	0.30	3.08	0.300	0.29	17.02	7.71	3.26	0.396
AMMMO	0.52	126.27	135.60	0.709	0.52	6.36	0.496	0.52	126.27	135.60	6.36	0.619

OCRL has a lower displayed J than the retrospective Best single algorithm under four of the six weights. At $(1,0,0)$, its compression rate is 0.21, compared with 0.23 for Best single algorithm. At the other three favorable weights, lower reported selection cost contributes to the J advantage. Best single selects retrospectively among fixed algorithms; OCRL makes a weight-conditioned path choice from the input. The exceptions are $(0,0,1)$ and $(0.5,0,0.5)$, where the single-algorithm reference has a lower J.

5.3 Effect of Weight-Conditioned Path Selection

We compare OCRL's weight-conditioned routes with the routes selected at $(0.4,0.3,0.3)$ on the same reported subset. The five weights use one frozen weight-by-type selector. Table 2 re-scores existing measurements using the same J accounting as Table 1. Both strategies include the same offline fitting charge, and relative reductions use unrounded scores. Weight-conditioned selection reduces J by 3.52% for compression time alone and 7.59% for decompression time alone. The reduction is 0.62% at $(0.5,0,0.5)$. At $(0.5,0.5,0)$, both strategies select identical routes; the small score difference comes from separate timing measurements. The $(0.4,0.3,0.3)$ row is a self-comparison. These results support the effectiveness of our model's weight-conditioned path selection for speed-oriented objectives in this subset.

Table 2: Objective values for weight-conditioned and fixed-$(0.4,0.3,0.3)$ routing on the post-hoc subset.

Weights (a,b,c)	OCRL J	Fixed-route J	Relative reduction
$(0,1,0)$	0.598	0.620	3.52%
$(0,0,1)$	0.293	0.317	7.59%
$(0.5,0.5,0)$	0.454	0.454	Same routes
$(0.5,0,0.5)$	0.300	0.302	0.62%
$(0.4,0.3,0.3)$	0.396	0.396	0.00%

6 Conclusion

Applications assign different priorities to compression rate, compression time, and decompression time. OCRL represents these priorities with adjustable objective weights. It searches reversible operator pipelines rather than fixing one compression algorithm pipeline. Its data- and weight-conditioned GRPO policy directs evaluations, while a certified byte bound prunes branches that cannot improve the best measured pipeline. These components address the three motivations: competing objectives, fixed pipelines, and dataset-dependent choices. In the reported subset, the measured OCRL routes have a lower displayed J than AMMMO under all six weights and than Best single algorithm under four weights.

AI USE STATEMENT

Generative AI tools helped polish the manuscript and monitor experiments run with our code. The authors supervised the experiments and reviewed the code changes and reported measurements. The authors made the scientific decisions and take responsibility for the paper.

ETHICS STATEMENT

This study evaluates lossless compression on existing numerical datasets and does not collect new participant data. Lossless compression preserves the input and does not provide privacy protection. Any redistribution of datasets or artifacts must respect the source licenses and access conditions.

REPRODUCIBILITY STATEMENT

The operator space, objective, and learning procedure are specified in Sections 3 and 4. Section 5.1 gives the data split, metrics, and timing rules; Appendix C gives operator examples. Our experimental records retain code versions, configurations, input identities, and raw measurements for the reported results. Archived baselines with different inputs or timing rules are identified in the comparison.

REFERENCES

Azim Afroozeh, Leonardo X. Kuffo, and Peter Boncz. ALP: Adaptive lossless floating-point compression. *Proceedings of the ACM on Management of Data*, 1(4):1–26, 2023. doi: 10.1145/3626717. URL https://doi.org/10.1145/3626717.

J. Alakuijala and Z. Szabadka. Brotli compressed data format. RFC 7932, RFC Editor, 2016. URL https://www.rfc-editor.org/rfc/rfc7932.html.

Vo Ngoc Anh and Alistair Moffat. Index compression using 64-bit words. *Software: Practice and Experience*, 40(2):131–147, 2010. doi: 10.1002/spe.948. URL https://doi.org/10.1002/spe.948.

Toygun Basaklar, Suat Gumussoy, and Ümit Y. Ogras. PD-MORL: preference-driven multi-objective reinforcement learning algorithm. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=zS9sRyaPFlJ.

Jon Louis Bentley, Daniel D. Sleator, Robert E. Tarjan, and Victor K. Wei. A locally adaptive data compression scheme. *Communications of the ACM*, 29(4):320–330, 1986. doi: 10.1145/5684.5688. URL https://www.cs.cmu.edu/~sleator/papers/adaptive-data-compression.pdf.

Davis Blalock, Samuel Madden, and John Guttag. Sprintz: Time series compression for the internet of things. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(3):1–23, 2018. doi: 10.1145/3264903. URL https://doi.org/10.1145/3264903.

M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. SRC Research Report 124, Digital Equipment Corporation, Systems Research Center, 1994. URL https://www.cs.jhu.edu/~langmea/resources/burrows_wheeler.pdf.

Martin Burtscher and Paruj Ratanaworabhan. FPC: A high-speed compressor for double-precision floating-point data. *IEEE Transactions on Computers*, 58(1):18–31, 2009. doi: 10.1109/TC.2008.131. URL https://userweb.cs.txstate.edu/~burtscher/papers/tc09.pdf.

Zhengxue Cheng, Yan Zhao, Keyu Wang, Hengdi Zhang, and Li Song. TaCo: A benchmark for lossless and lossy codecs of heterogeneous tactile data. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/pdf?id=1PYXFkS6Hy.

Yann Collet. Zstandard compression and the 'application/zstd' media type. RFC 8878, RFC Editor, 2021. URL https://www.rfc-editor.org/rfc/rfc8878.html.

P. Deutsch. DEFLATE compressed data format specification version 1.3. RFC 1951, RFC Editor, 1996. URL https://www.rfc-editor.org/rfc/rfc1951.html.

Jarek Duda. Asymmetric numeral systems: Entropy coding combining speed of Huffman coding with compression rate of arithmetic coding, 2013. URL https://arxiv.org/abs/1311.2540.

Mohamad Khalil Farhat, Ji Zhang, Xiaohui Tao, and Tianning Li. DICOMP: Deep reinforcement learning for integer compression. *Machine Learning with Applications*, 22:100756, 2025. doi: 10.1016/j.mlwa.2025.100756. URL https://doi.org/10.1016/j.mlwa.2025.100756.

David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952. doi: 10.1109/JRPROC.1952.273898. URL https://doi.org/10.1109/JRPROC.1952.273898.

Maximilian Kuschewski, David Sauerwein, Adnan Alhomssi, and Viktor Leis. BtrBlocks: Efficient columnar compression for data lakes. *Proceedings of the ACM on Management of Data*, 1(2):1–26, 2023. doi: 10.1145/3589263. URL https://doi.org/10.1145/3589263.

Daniel Lemire and Leonid Boytsov. Decoding billions of integers per second through vectorization. *Software: Practice and Experience*, 45(1):1–29, 2015. doi: 10.1002/spe.2203. URL https://doi.org/10.1002/spe.2203.

Ruiyuan Li, Zheng Li, Yi Wu, Chao Chen, and Yu Zheng. Elf: Erasing-based lossless floating-point compression. *Proceedings of the VLDB Endowment*, 16(7):1763–1776, 2023. doi: 10.14778/3587136.3587149. URL https://www.vldb.org/pvldb/vol16/p1763-li.pdf.

Panagiotis Liakos, Katia Papakonstantinopoulou, and Yannis Kotidis. Chimp: Efficient lossless floating point compression for time series databases. *Proceedings of the VLDB Endowment*, 15(11):3058–3070, 2022. doi: 10.14778/3551793.3551852. URL https://vldb.org/pvldb/vol15/p3058-liakos.pdf.

Anji Liu, Stephan Mandt, and Guy Van den Broeck. Lossless compression with probabilistic circuits. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=X_hByk2-5je.

Tuomas Pelkonen, Scott Franklin, Justin Teller, Paul Cavallaro, Qi Huang, Justin Meza, and Kaushik Veeraraghavan. Gorilla: A fast, scalable, in-memory time series database. *Proceedings of the VLDB Endowment*, 8(12):1816–1827, 2015. doi: 10.14778/2824032.2824078. URL https://www.vldb.org/pvldb/vol8/p1816-teller.pdf.

Andrew Rodriguez, Noushin Azami, and Martin Burtscher. Adaptive per-file lossless compression of floating-point data. In *2024 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 423–430. IEEE, 2024. doi: 10.1109/IPDPSW63119.2024.00092. URL https://doi.org/10.1109/IPDPSW63119.2024.00092.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300, 2024. doi: 10.48550/ARXIV.2402.03300. URL https://doi.org/10.48550/arXiv.2402.03300.

Yuxin Tang, Feng Zhang, Jiawei Guan, Yuan Tian, Xiangdong Huang, Chen Wang, Jianmin Wang, and Xiaoyong Du. Improving time series data compression in Apache IoTDB. *Proceedings of the VLDB Endowment*, 18(10):3406–3420, 2025. doi: 10.14778/3748191.3748204. URL https://doi.org/10.14778/3748191.3748204.

James Townsend, Thomas Bird, and David Barber. Practical lossless compression with latent variables using bits back coding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=ryE98iR5tm.

Tuan Truong, Rithwik Sudharsan, Yibo Yang, Peter Xiangyuan Ma, Ruihan Yang, Stephan Mandt, and Joshua S. Bloom. AstroCompress: A benchmark dataset for multi-purpose compression of astronomical data. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=kQCHCkNk7s.

Chen Wang, Jialin Qiao, Xiangdong Huang, Shaoxu Song, Haonan Hou, Tian Jiang, Lei Rui, Jianmin Wang, and Jiaguang Sun. Apache IoTDB: A time series database for IoT applications. *Proceedings of the ACM on Management of Data*, 1(2):1–27, 2023. doi: 10.1145/3589775. URL https://doi.org/10.1145/3589775.

Ian H. Witten, Radford M. Neal, and John G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987. doi: 10.1145/214762.214771. URL https://doi.org/10.1145/214762.214771.

Jinzhao Xiao, Yuxiang Huang, Changyu Hu, Shaoxu Song, Xiangdong Huang, and Jianmin Wang. Time series data encoding for efficient storage: A comparative analysis in Apache IoTDB. *Proceedings of the VLDB Endowment*, 15(10):2148–2160, 2022. doi: 10.14778/3547305.3547319. URL https://www.vldb.org/pvldb/vol15/p2148-song.pdf.

Jinzhao Xiao, Zihan Guo, and Shaoxu Song. BOS: Bit-packing with outlier separation. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*, pp. 1375–1387. IEEE, 2025. doi: 10.1109/ICDE65448.2025.00107. URL https://doi.org/10.1109/ICDE65448.2025.00107.

Zeyu Yang, Tianyi Zhang, Jianwen Xie, Chuan Li, Zhaozhuo Xu, and Anshumali Shrivastava. To compress or not? pushing the frontier of lossless GenAI model weights compression with exponent concentration. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/57c3fef21638eb9d1d67d32ba0a6d1af-Abstract-Conference.html.

Xinyang Yu, Yanqing Peng, Feifei Li, Sheng Wang, Xiaowei Shen, Huijun Mai, and Yue Xie. Two-level data compression using machine learning in time series database. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, 2020. doi: 10.1109/ICDE48307.2020.00119. URL https://users.cs.utah.edu/~lifeifei/papers/MLCompression.pdf.

Guoying Zhu, Haipeng Dai, Kang Yuan, Qian Wang, Lida Chen, Zhenghong Luo, Meng Li, Rong Gu, Xizi Ni, Hua Fan, Dachao Fu, and Wenchao Zhou. Local-to-cloud database synchronization via fine-grained hybrid compression. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*, pp. 2977–2989. IEEE, 2025. doi: 10.1109/ICDE65448.2025.00223. URL https://doi.org/10.1109/ICDE65448.2025.00223.

J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977. doi: 10.1109/TIT.1977.1055714. URL https://doi.org/10.1109/TIT.1977.1055714.

A Compression Algorithms Displayed in Figure 1

The comparison uses archived method labels with eligible results on every dataset in the plotted collection. For each method, the metrics are averaged equally over that collection. The two figures rank these means separately, using compression rate and either compression time or decompression time. A label is Pareto-nondominated if no other eligible label is at least as good on both metrics and strictly better on one. Successive ranks remove the preceding non-dominated set before repeating this comparison. Together, the Top-10 sets in Figures 1(a) and 1(b) contain 14 distinct algorithm labels, listed in Table 3. The strict first Pareto front has one label in Figure 1(a) and five in Figure 1(b). Later ranks fill each Top-10 and can include labels dominated by earlier ranks. For a partially included rank, descending NSGA-II crowding distance and then alphabetical label order determine the cutoff. Archived timing protocols differ, so these ranks illustrate reported trade-offs rather than a controlled speed comparison. These labels denote archived project implementations or presets. They do not imply that every measured row is byte-identical to an upstream release. The mapping from each plotted label to its archived keys is retained in `figures/motivation_common18_20260926/selected_method_mapping.csv`. In particular, BOS denotes the archived `ts_2diff_bos` adapter. It selects between two candidates per column rather than executing the full outlier-separation BOS pipeline. Several other labels also select a preset by minimum reported size within each archived dataset and type record.

Table 3: Algorithm labels used by the shared legend in Figure 1. "Artifact source" marks a project implementation label for which we do not claim a byte-identical correspondence to an external algorithm.

Label	Archived key	Mechanism represented by the plotted label	Reference
2DIFF	`TS_2DIFF`	Uses blockwise adjacent deltas, a minimum-delta frame of reference, and fixed-width packing. The floating-point path first applies reversible decimal scaling.	Xiao et al. (2022)
ALP	`alp`	Adaptive decimal scaling converts exactly representable values to integers, followed by frame-of-reference packing with explicit exceptions or raw fallbacks.	Afroozeh et al. (2023)
BOS	`ts_2diff_bos`	Compares TS_2DIFF with a BOS64 adapter for each column and reports the smaller payload. The adapter uses adjacent delta, ZigZag, and fixed-width packing after reversible decimal scaling for floats. It is not the full outlier-separation BOS pipeline.	Artifact source; backgrounds in (Xiao et al., 2022; 2025)
Chimp128	`CHIMP128`	Uses the archived numeric-word Chimp128 profile; upstream wire identity is not asserted.	Liakos et al. (2022)
Dict.	`col_dictionary`	Builds a first-seen dictionary of 64-bit values (exact bit patterns for floating point) and replaces the input with 32-bit dictionary indices.	Artifact source
FastPFor	`fastpfor`	Patched frame-of-reference coding bit-packs regular integers and records exceptions separately. The project profile includes a VariableByte tail.	Lemire & Boytsov (2015)
Lizard	`lizard_*`	Archived Lizard presets under the same reported-size selection rule.	Artifact source
LZ4 Fast	`lz4fast_*`	Archived LZ4 fast-mode presets; the reported-size-minimizing available preset is selected per dataset/type record.	Artifact source
LZAV	`lzav_*`	Two archived LZAV presets under the same reported-size selection rule.	Artifact source
LZHAM	`lzham_*`	Archived LZHAM presets under the same reported-size selection rule.	Artifact source
LZVN	`lzvn`	The archived LZVN compression implementation.	Artifact source
Tornado	`tornado_*`	Archived Tornado presets under the same reported-size selection rule.	Artifact source
XPACK	`xpack`	The archived XPACK compression implementation.	Artifact source
XZ	`xz_level_0`–`xz_level_9`	Archived XZ compression levels under the same reported-size selection rule.	Artifact source

The descriptions above explain the plotted implementation labels. They do not redefine the underlying algorithms. All displayed labels have dataset-level results for the same collection. Bird-migration still contributes only its two integer columns. The plotted values are archived reported-output measurements, not audited complete frames.

B PRUNING PROOFS AND STORAGE-ONLY DOMINANCE

This appendix proves Propositions 4.1 and 4.2, stated in Section 4.2. The separate storage-only dominance condition is given below. For that condition, write $B_{\mathrm{frame}}(p) = |C_p(x)|$ and let U_B be the smallest verified complete-frame size.

Proof of Proposition 4.1. The frame must retain B_{fix} and all records counted in $B_{\mathrm{keep}}(s)$. A leaf at depth K cannot receive another ordinary operator. Each counted b_ℓ is certified to survive routing, STOP, and final serialization; a flow without such a certificate contributes zero. The counted bits are disjoint, so every completion retains their sum. These bits require at least the corresponding rounded-up byte count even if routing shares padding. The separately retained overhead and metadata contribute $B_{\mathrm{fix}} + B_{\mathrm{keep}}(s)$ bytes. Other payloads and future metadata are nonnegative, which proves the bound. If $L(s) > U_B$, every completion exceeds the verified incumbent's frame size. □

Proof of Proposition 4.2. By Proposition 4.1, $\rho \geq L(s)/|x|$. Both normalized time costs in (2) are nonnegative. The nonnegative weighted sum of these lower bounds is $aL(s)/|x|$, which proves the claim. If $\underline{J}_w(s) > U_J$, every completion exceeds the verified incumbent's cost. □

Proposition B.1 (Dominance of equivalent continuation states)**.** *Suppose two reversible prefixes p_1, p_2 have the same continuation state z and legal continuation set $\mathcal{Q}(z)$. Let $B_{\mathrm{pre},i}$ be the fixed byte cost of retaining prefix p_i. Let $C_{\mathrm{rem}}(z, \xi)$ be the remaining byte cost of continuation ξ, shared by both prefixes. Assume their complete sizes decompose as*

$$B_{\mathrm{frame}}(p_i\xi) = B_{\mathrm{pre},i} + C_{\mathrm{rem}}(z, \xi), \qquad i \in \{1, 2\}, \quad \xi \in \mathcal{Q}(z). \tag{8}$$

If $B_{\mathrm{pre},1} \leq B_{\mathrm{pre},2}$, retaining and exploring p_1 while discarding p_2 preserves the minimum attainable storage size.

Proof. Every completion of p_2 has a legal counterpart through p_1, and

$$B_{\mathrm{frame}}(p_2\xi) - B_{\mathrm{frame}}(p_1\xi) = B_{\mathrm{pre},2} - B_{\mathrm{pre},1} \geq 0. \tag{9}$$

Thus no discarded completion improves on its retained counterpart. □

The continuation state z must contain every field that affects future legality or C_{rem}. These fields include the ordered active and stopped data flows, their parent ownership, identifiers, bytes, types, and layouts. They also include the remaining depth budgets and total decisions $H - t$. The reconstruction context and any history-dependent serialization state must also be included. Equal payload sizes alone are insufficient. Proposition B.1 permits duplicate-state or execution-order elimination only under one condition. The decomposition in (8) must hold for the frame format.

Storage dominance in Proposition B.1 does not establish dominance under an objective with time terms: a larger frame may encode or decode faster.

C Operators and Reversible Examples

Table 4 lists $O_1, \ldots, O_{30}$ in consecutive order within eight operator groups. An application of O_i acts on the ordered input tuple $\mathcal{I}_t$ and the required context. The tuple contains typed data flows $F = (\tau, n, \mathbf{v}, \beta)$. The operator produces output data flows and reconstruction information. The examples describe mechanisms, not measured packets or compression gains. All stated side information belongs to the complete representation. Each O_i denotes an operator, not a guarantee of an independently executable kernel. The implementation directory also distinguishes shared mechanisms, configuration operators, reference proxies, and composite operators. The three special operations STOP, CONCAT, and SPLIT (including continuous slicing) remain outside these 30 operators.

Table 4: The 30 operators organized into eight operator groups, with their identifiers and names.

No.	Operator group	Operators
1	Bit processing	O_1: Eraser; O_2: Integer Bit Packing; O_3: Trailing-Zero Packing; O_4: Integer Coding; O_5: Bitstream Packing; O_6: Column Split
2	Layout and state	O_7: Block-State Control; O_8: Container Framing
3	LZ decisions and format	O_9: Match Parsing Policy; O_{10}: Match Window Model; O_{11}: Match Token Format; O_{12}: Literal Entropy Coding
4	Match and repetition	O_{13}: Dictionary Coding; O_{14}: Run-Length Encoding; O_{15}: LZ Matching
5	Numeric representation	O_{16}: Scaling; O_{17}: Frame of Reference; O_{18}: Outlier Separation; O_{19}: XOR; O_{20}: ZigZag
6	Probability coding	O_{21}: Huffman Coding; O_{22}: Range/Arithmetic Coding; O_{23}: FSE/rANS Coding; O_{24}: Specialized Entropy Coding; O_{25}: Probability Prediction
7	Reordering and symbol recoding	O_{26}: Value Reordering; O_{27}: Block Transform; O_{28}: Symbol Recoding
8	Residual formation	O_{29}: Regression Prediction; O_{30}: Delta

C.1 Bit Processing

O_1**: Eraser.** Eraser removes or recodes low bits while preserving exact reconstruction. In a retained-bit example, removing four low bits from $45 = 16 \cdot 2 + 13$ gives main value 2 and side value 13. The split width restores 45 exactly. Decimal-aware variants instead retain decimal reconstruction parameters and literal exceptions when needed.

O_2**: Integer Bit Packing.** Bit packing stores unsigned values in a selected width: $(1, 2, 3)$ becomes $01\,10\,11$ at two bits per value. The decoder needs the width, value count, bit order, and padding convention. Choosing a width never discards significant bits.

O_3**: Trailing-Zero Packing.** Common zero bits or zero regions of residual words can be omitted with their positions recorded. For $(24, 40)$, removing three shared trailing zeros gives $(3, 5)$ and shift 3. Left shifts restore the input. Window-based variants additionally retain window widths and reuse tags.

O_4**: Integer Coding.** Variable-length integer codes allocate more storage to larger unsigned values. For example, unsigned little-endian base-128 (ULEB128) coding maps 300 to hexadecimal bytes `AC 02`. After stripping the continuation bit, $44 + 2 \cdot 128 = 300$. Signed values require a separate reversible mapping.

O_5**: Bitstream Packing.** This operator serializes fields with specified bit widths and order. Fields 101 and 11, of widths 3 and 2, give 10111. The padded most-significant-bit-first byte is 10111000. Retaining the field layout and five valid bits removes ambiguity.

O_6**: Column Split.** Column split decomposes records or bit fields into separately encodable data flows. For example, width-two byte records $[01\ 02][03\ 04]$ become columns $[01\ 03]$ and $[02\ 04]$. Record width, row count, and column order restore interleaving. Float deconstruction, bit-field splitting, and bit unpacking similarly retain the exact bit layout and valid-bit information. This operator differs from continuous SPLIT, which partitions consecutive positions.

C.2 LAYOUT AND STATE

O_7**: Block-State Control.** Block-state control specifies boundaries and how decoder state is initialized or carried between blocks. For independently delta-coded blocks $(10, 11) \mid (50, 51)$, resetting at the boundary gives anchors $(10, 50)$ and residuals $(1) \mid (1)$. Reconstruction requires both block lengths and the reset convention. This example illustrates state control. Implementation entries associated with this operator also include composite operators.

O_8**: Container Framing.** Framing serializes payloads and the information needed to identify and delimit them. An elementary envelope $[\mathrm{magic}, \mathrm{length} = 3, \texttt{ABC}]$ lets the decoder validate the format and recover the three payload bytes. A complete pipeline also retains its types, tree structure, and reconstruction metadata.

C.3 LZ DECISIONS AND FORMAT

O_9**: Match Parsing Policy.** A parsing policy selects which literals and available matches form the encoded sequence. For `ABCDEFGHABCDEFGH`, an aligned-eight greedy parse emits eight literals followed by a copy of distance 8 and length 8. The selected token sequence, rather than an encoder search trace, determines reconstruction.

O_{10}**: Match Window Model.** The window limits which prior positions may supply a match. A candidate at distance 8 is allowed by an eight-byte window but excluded by a four-byte window. The selected window and any history-reset rules must be reproducible during decoding. Window configuration alone does not perform match search.

O_{11}**: Match Token Format.** Token formats serialize the literal or match decisions made by earlier operators. For a copy of distance 3 and length 3, a format may store a match flag followed by 32-bit distance and length fields. Field widths, byte order, token count, and any length bias define the inverse parser.

O_{12}**: Literal Entropy Coding.** Literal coding compresses the literal subsequence while preserving the match records needed to place it. With codewords $A : 0$ and $B : 1$, literals `ABA` become 010. The codebook, literal count, and untouched match-distance and match-length data flows recover their original positions.

C.4 MATCH AND REPETITION

O_{13}**: Dictionary Coding.** Dictionary coding replaces repeated values with identifiers. Values $(7, 7, 9, 7)$ become identifiers $(0, 0, 1, 0)$ under dictionary $(7, 9)$. Retaining the dictionary and identifier order restores the values.

O_{14}**: Run-Length Encoding.** RLE represents consecutive equal values through a value data flow and a run-length data flow. For $(7, 7, 7, 9, 9)$, these data flows are $(7, 9)$ and $(3, 2)$. Corresponding positions identify each run, and expansion restores five values. The illustrated run representation is one mode of implementations that may also select packed or raw blocks.

O_{15}**: LZ Matching.** LZ matching finds repeated substrings that can be reconstructed from earlier output. For `ABCABC`, literals `ABC` followed by $\mathrm{copy}(3, 3)$ reproduce the sequence, where the arguments denote distance and length. The token convention specifies how copies, including overlapping copies, are expanded.

C.5 Numeric Representation

O_{16}**: Scaling.** Scaling represents suitable floating-point values as integers using recorded scale parameters. For $(1.25, 1.50)$, scale 100 gives $(125, 150)$. Inverse division restores the original floating-point bits. Values that fail exact reconstruction require exceptions or a literal representation.

O_{17}**: Frame of Reference.** FOR subtracts a shared base to reduce the range of a block. Input $(100, 103, 101)$ gives base 100 and offsets $(0, 3, 1)$. Adding the retained base with the specified word arithmetic restores every value.

O_{18}**: Outlier Separation.** Outlier separation keeps a regular data flow and information that restores exceptional values. For $(1, 2, 2, 3, 100)$, clipping to a declared regular range $[0, 5]$ produces $(1, 2, 2, 3, 5)$ and the zero-based exception record $(4, 100)$. Replacing that position recovers the input.

O_{19}**: XOR.** XOR represents a word relative to a recorded or reproducible reference word. Floating-point words `3ff0000000000000` and `3ff8000000000000` give residual `0008000000000000`. Retaining the first word and XORing it with the residual restores the second word exactly.

O_{20}**: ZigZag.** ZigZag maps signed integers near zero to small unsigned integers: $(-2, -1, 0, 1, 2) \mapsto (3, 1, 0, 2, 4)$. Code parity identifies the sign, and the declared word width fixes the inverse. This mapping does not itself pack the resulting integers.

C.6 Probability Coding

O_{21}**: Huffman Coding.** Huffman coding assigns prefix-free codewords from symbol frequencies. Using $A : 0$, $B : 10$, and $C : 11$, `ABAC` becomes 010011. The codebook, symbol count, and padding information make decoding unambiguous.

O_{22}**: Range/Arithmetic Coding.** Range coding repeatedly narrows an interval according to symbol probabilities. With symbol probabilities $\Pr(A) = 1/2$ and $\Pr(B) = \Pr(C) = 1/4$ in alphabet order, `AB` narrows $[0, 1)$ to $[1/4, 3/8)$. A code value in that interval, the probability model, and the message length identify the sequence under the agreed arithmetic convention.

O_{23}**: FSE/rANS Coding.** Finite-state entropy (FSE) and range asymmetric numeral systems (rANS) encode symbols through reversible integer-state transitions. This example uses an rANS core. Consider a total frequency of 4, with symbol A having frequency 3 and cumulative frequency 0. Encoding A maps state 4 to $\lfloor 4/3 \rfloor 4 + (4 \bmod 3) = 5$. Decoding returns $3\lfloor 5/4 \rfloor + (5 \bmod 4) = 4$. The frequency table, symbol count, final state, and normalization bytes are required, with normalization omitted from this single-transition example.

O_{24}**: Specialized Entropy Coding.** This historical operator includes specialized representations such as SPDP's context-repeat operator, which is distinct from Huffman, range, and ANS coding. Suppose a previous six-byte context `abcdef` was followed by `ABY`. Another occurrence followed by `ABZ` can emit repeat count 2 and literal `Z`. Matching history initialization, updates, and the output length allow reconstruction.

O_{25}**: Probability Prediction.** A probability prediction operator uses a causal model to supply probabilities to a compatible entropy coding operator. It does not produce a compressed payload or a numeric residual. For example, a model can assign probability $\Pr(1 \mid h_t) = 3/4$ from previously decoded context h_t. The encoder and decoder process the current bit using that probability before observing it and updating their identical states. Initialization and reset rules must be reconstructible, and an unconnected probability prediction operator is not a complete compression path.

C.7 Reordering and Symbol Recoding

O_{26}**: Value Reordering.** Value reordering changes sequence order to expose regularity while retaining the inverse permutation. Stable sorting $(3, 1, 3, 2)$ yields $(1, 2, 3, 3)$ and zero-based source indices $(1, 3, 0, 2)$. Placing each sorted value at its recorded index restores the input.

O_{27}**: Block Transform.** Block transform operators reorganize values or bytes within a declared block. For example, the cyclic Burrows–Wheeler transform operator maps `banana` to `nnbaaa`, with zero-based primary row 3. The block length and primary row identify the original sequence through the inverse operator.

O_{28}**: Symbol Recoding.** Symbol recoding changes symbol identifiers according to a reconstructible ordering or context. Move-to-front coding with initial byte alphabet $(0, 1, \ldots, 255)$ maps $(2, 2, 1)$ to positions $(2, 0, 2)$. Moving each recovered symbol to the front recreates the same evolving alphabet.

C.8 Residual Formation

O_{29}**: Regression Prediction.** Regression prediction stores errors relative to a numeric model. For $(10, 12, 14)$ and zero-based positions i, predictor $10 + 2i$ gives residuals $(0, 0, 0)$. The coefficients, block length, rounding convention, and residuals restore every integer before any subsequent signed mapping.

O_{30}**: Delta.** Delta forms differences from a specified earlier value or predictor. The adjacent-difference example $(10, 13, 17)$ gives anchor 10 and residuals $(3, 4)$. Cumulative addition restores the sequence. Lagged and higher-order variants require their corresponding anchors and predictor settings.